A Markov Chain Model Checker

Holger Hermanns[1], Joost-Pieter Katoen[1],
Joachim Meyer-Kayser[2⋆], and Markus Siegle[2]

P.O. Box 217, 7500 AE Enschede, The Netherlands
[2] Lehrstuhl für Informatik 7, University of Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen, Germany

Abstract. Markov chains are widely used in the context of performance and reliability evaluation of systems of various nature. Model checking of such chains with respect to a given (branching) temporal logic formula has been proposed for both the discrete [17,6] and the continuous time setting [4,8]. In this paper, we describe a prototype model checker for discrete and continuous-time Markov chains, the *Erlangen–Twente Markov Chain Checker* ($\mathsf{E} \vdash MC^2$), where properties are expressed in appropriate extensions of CTL. We illustrate the general benefits of this approach and discuss the structure of the tool. Furthermore we report on first successful applications of the tool to non-trivial examples, highlighting lessons learned during development and application of $\mathsf{E} \vdash MC^2$.

1 Introduction

Markov chains are widely used as simple yet adequate models in diverse areas, ranging from mathematics and computer science to other disciplines such as operations research, industrial engineering, biology and demographics. Markov chains can be used to estimate performance characteristics of various nature, for instance to quantify throughput of manufacturing systems, locate bottlenecks in communication systems, or to estimate reliability in aerospace systems.

Model checking is a very successful technique to establish the *correctness* of systems from similar application domains, usually described in terms of a non-deterministic finite-state model. If non-determinism is replaced by randomized, i.e. probabilistic decisions, the resulting model boils down to a finite-state discrete-time Markov chain (DTMC). For these models, qualitative and quantitative model checking algorithms have been investigated extensively, see e.g. [3,5,6,10,13,17,29]. In a qualitative setting it is checked whether a property holds with probability 0 or 1; in a quantitative setting it is verified whether the probability for a certain property meets a given lower- or upper-bound.

Markov chains are *memoryless*. In the discrete-time setting this is reflected by the fact that probabilistic decisions do not depend on the outcome of decisions taken earlier, only the state currently occupied is decisive to completely

⋆ supported by the German Research Council DFG under HE 1408/6-1.

S. Graf and M. Schwartzbach (Eds.): TACAS/ETAPS 2000, LNCS 1785, pp. 347–362, 2000.

determine the probability of next transitions. For continuous-time Markov chains (CTMCs), where time ranges over (positive) reals (instead of discrete subsets thereof) the memoryless property further implies that probabilities of taking next transitions do not depend on the amount of time spent in the current state. The vast majority of applications of Markov chain modelling involves CTMCs, as opposed to DTMCs.[1] In particular, CTMCs are the underlying semantic model of major high-level performance modelling formalisms such as stochastic Petri nets [1], stochastic automata networks [26], stochastic process algebras [24,21], Markovian queueing networks [12], and various extensions thereof.

Model checking of CTMCs has been discussed in [8], introducing a (branching) temporal logic called *continuous-time stochastic logic* (**CSL**) to express properties over CTMCs. This logic is an extension of the (equally named) logic by Aziz et al. [4] with an operator to reason about steady-state probabilities: e.g. the formula $\mathcal{S}_{\geqslant p}(\Phi)$ asserts that the steady-state probability for being in a Φ-state is at least p, for $p \in [0,1]$. Apart from the usual quantifiers like next and until, a time-bounded until $\mathcal{U}^{\leqslant t}$, for t a non-negative real, is incorporated, for which standard derivatives, such as a time-bounded eventually $\Diamond^{\leqslant t}$, can be defined. The usual path quantifiers $\forall$ and $\exists$ are replaced by the probabilistic operator $\mathcal{P}_{\bowtie p}(.)$ for comparison operator $\bowtie$ and $p \in [0,1]$. For instance, $\mathcal{P}_{<10^{-9}}(\Diamond^{\leqslant 4} \mathit{error})$ asserts that the probability for a system error within 4 time-units is less than 10^{-9}. Such properties are out of the scope of what can be computed with standard Markov chain analysis algorithms, yet they are highly interesting to study.

In this paper we describe the *Erlangen–Twente Markov Chain Checker* ($\mathsf{E} \vdash MC^2$), to our knowledge the first implementation of a model checker for CTMCs. It uses numerical methods to model check **CSL**-formulas, based on [8]. Apart from standard graph algorithms, model checking **CSL** involves matrix-vector multiplications (for next-formulas), solutions of linear systems of equations (for until- and steady-state formulas), and solutions of systems of Volterra integral equations (for time-bounded until). Linear systems of equations are iteratively solved by standard numerical methods [27]. Systems of integral equations are iteratively solved by piecewise integration after discretization. As a side result, $\mathsf{E} \vdash MC^2$ is also capable to model check DTMCs against properties expressed in **PCTL** [17]. This is not surprising, taking into account that the algorithms needed for **CSL** are a superset of what is needed to check **PCTL**. The tool has been implemented in Java (version 1.2), and uses sparse matrix representations. The paper illustrates how $\mathsf{E} \vdash MC^2$ can be linked (among others) to (generalized) stochastic Petri nets (GSPN) and to Markovian queueing networks by reporting on the model checking of a GSPN-model of a cyclic server system and of a tandem queueing network.

Other model checkers for probabilistic systems are the DTMC-model checkers ProbVerus [18] and TPWB (the Timing and Probability Work-Bench) [15],

[1] DTMCs are mostly applied in strictly synchronous scenarios, while CTMCs have shown to fit well to (interleaving) asynchronous scenarios.

and the recent symbolic model checker for (discrete-time) Markov decision processes [2].

The paper is organized as follows. Section 2 briefly introduces CTMCs and **CSL**. Section 3 discusses the tool architecture together with the model checking algorithm and some implementation details. Section 4 reports on practical experiences with two case studies and Section 5 concludes the paper.

2 Continuous-Time Markov Chains and CSL

This section reviews continuous-time Markov chains [27] and **CSL** [8].

Continuous-time Markov chains. Let AP be a fixed, finite set of atomic propositions. A (labelled) *continuous-time Markov chain* (CTMC for short) is a tuple $\mathcal{M} = (S, \mathbf{R}, L)$ where S is a finite set of *states*, $\mathbf{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ the *rate matrix*, and $L : S \to 2^{AP}$ the *labelling* function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions $a \in AP$ that are valid in s.

Intuitively, $\mathbf{R}(s, s')$ specifies that the probability of moving from state s to s' within t time-units (for positive t) is $1 - e^{-\mathbf{R}(s,s')\cdot t}$, an exponential distribution with rate $\mathbf{R}(s, s')$. If $\mathbf{R}(s, s') > 0$ for more than one state s', a competition between the transitions exists, known as the *race condition*. Let $\mathbf{E}(s) = \sum_{s' \in S} \mathbf{R}(s, s')$, the total rate at which any transition emanating from state s is taken. This rate is the reciprocal of the mean sojourn time in s. More precisely, $\mathbf{E}(s)$ specifies that the probability of leaving s within t time-units (for positive t) is $1 - e^{-\mathbf{E}(s)\cdot t}$, due to the fact that the minimum of exponential distributions (competing in a race) is again exponentially distributed, and characterized by the sum of their rates. Consequently, the probability of moving from state s to s' by a single transition, denoted $\mathbf{P}(s, s')$, is determined by the probability that the delay of going from s to s' finishes before the delays of other outgoing edges from s; formally, $\mathbf{P}(s, s') = \mathbf{R}(s, s')/\mathbf{E}(s)$ (except if s is an absorbing state, i.e. if $\mathbf{E}(s) = 0$; in this case we define $\mathbf{P}(s, s') = 0$). Remark that the matrix $\mathbf{P}$ describes an embedded DTMC.

A *path* σ is a finite or infinite sequence $s_0, t_0, s_1, t_1, s_2, t_2, \ldots$ with for $i \in \mathbb{N}$, $s_i \in S$ and $t_i \in \mathbb{R}_{>0}$ such that $\mathbf{R}(s_i, s_{i+1}) > 0$, if σ is infinite. For infinite path σ, $t \in \mathbb{R}_{\geqslant 0}$ and i the smallest index with $t \leqslant \sum_{j=0}^{i} t_j$ let $\sigma@t = s_i$, the state of σ at time t. If σ is finite and ends in s_l, we require that s_l is absorbing, and $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $i < l$. For finite σ, $\sigma@t = s_l$ for $t > \sum_{j=0}^{l-1} t_j$, for other t it is defined as above. Let $Path(s)$ be the set of paths starting in s.

Continuous stochastic logic. CSL is a branching-time, CTL-like temporal logic where the state-formulas are interpreted over states of a CTMC.

Definition 1. *For $a \in AP$, $p \in [0, 1]$ and $\bowtie \ \in \{\leqslant, <, \geqslant, >\}$, the state-formulas of* **CSL** *are defined by the grammar*

$$\Phi ::= a \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg\Phi \;\Big|\; \mathcal{P}_{\bowtie p}(X\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\Phi\,\mathcal{U}\,\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\Phi\,\mathcal{U}^{\leqslant t}\,\Phi) \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi)$$

The other boolean connectives are derived in the usual way. The probabilistic operator $\mathcal{P}_{\bowtie p}(.)$ replaces the usual CTL path quantifiers $\exists$ and $\forall$ that can be re-invented – up to fairness [9] – as the extremal probabilities $\mathcal{P}_{>0}(.)$ and $\mathcal{P}_{\geqslant 1}(.)$. Formula $\mathcal{P}_{\bowtie p}(X\Phi)$ ($\mathcal{P}_{\bowtie p}(\Phi_1\,\mathcal{U}\,\Phi_2)$, respectively) asserts that the probability measure of the paths satisfying $X\Phi$ ($\Phi_1\,\mathcal{U}\,\Phi_2$) satisfies $\bowtie p$. The meaning of X ("next step") and $\mathcal{U}$ ("until") is standard. The temporal operator $\mathcal{U}^{\leqslant t}$ is the real-time variant of $\mathcal{U}$; $\Phi_1\,\mathcal{U}^{\leqslant t}\,\Phi_2$ asserts that $\Phi_1\,\mathcal{U}\,\Phi_2$ will be satisfied in the time interval $[0,t]$; i.e. there is some $x \in [0,t]$ such that Φ_1 continuously holds during the interval $[0,x[$ and Φ_2 becomes true at time instant x. The state formula $\mathcal{S}_{\bowtie p}(\Phi)$ asserts that the steady-state probability for a Φ-state satisfies $\bowtie p$. Temporal operators like $\Diamond$, $\Box$ and their real-time variants $\Diamond^{\leqslant t}$ or $\Box^{\leqslant t}$ can be derived, e.g. $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t}\,\Phi) = \mathcal{P}_{\bowtie p}(\mathit{true}\,\mathcal{U}^{\leqslant t}\,\Phi)$ and $\mathcal{P}_{\geqslant p}(\Box\,\Phi) = \mathcal{P}_{\leqslant 1-p}(\Diamond\,\neg\Phi)$.

Semantics of CSL. Let $\mathcal{M} = (S, \mathbf{R}, L)$ with proposition labels in AP. The semantics for atomic propositions, negation, and conjunction is standard [11]. Let $\mathit{Sat}(\Phi) = \{\, s \in S \mid s \models \Phi \,\}$. The steady-state operator is defined by:

$$s \models \mathcal{S}_{\bowtie p}(\Phi) \quad \text{iff} \quad \pi_{\mathit{Sat}(\Phi)}(s) \bowtie p$$

where $\pi_{S'}(s)$ denotes the steady-state probability for $S' \subseteq S$ when starting in s,

$$\pi_{S'}(s) \;=\; \lim_{t\to\infty} \;\Pr\{\, \sigma \in \mathit{Path}(s) \mid \sigma@t \in S' \,\}.$$

This limit exists for finite S [27]. Obviously, $\pi_{S'}(s) = \sum_{s'\in S'} \pi_{s'}(s)$, where we write $\pi_{s'}(s)$ instead of $\pi_{\{\,s'\,\}}(s)$. We let $\pi_{\varnothing}(s) = 0$.

For path-formula φ of the form $X\Phi$, $\Phi_1\,\mathcal{U}\,\Phi_2$, or $\Phi_1\,\mathcal{U}^{\leqslant t}\,\Phi_2$ we have:

$$s \models \mathcal{P}_{\bowtie p}(\varphi) \;\text{ iff }\; \mathit{Prob}(s,\varphi) \bowtie p, \quad \text{where } \mathit{Prob}(s,\varphi) = \Pr\{\, \sigma \in \mathit{Path}(s) \mid \sigma \models \varphi \,\}$$

i.e., $\mathit{Prob}(s,\varphi)$ denotes the probability measure of all paths $\sigma \in \mathit{Path}(s)$ satisfying φ The fact that, for each state s, the set $\{\sigma \in \mathit{Path}(s) \mid \sigma \models \varphi\}$ is measurable, follows by easy verification given the Borel space construction on paths through CTMCs in [8]. The semantics of next and until-formulas is standard [11] and is omitted here. For time-bounded until we have:

$$\sigma \models \Phi_1\,\mathcal{U}^{\leqslant t}\,\Phi_2 \qquad \text{iff} \qquad \exists x \in [0,t].\; (\sigma@x \models \Phi_2 \wedge \forall y \in [0,x[.\; \sigma@y \models \Phi_1)\,.$$

3 The Model Checker $\mathsf{E} \vdash MC^2$

$\mathsf{E} \vdash MC^2$ is a prototype tool supporting the verification of **CSL**-properties over CTMCs. It is a *global* model checker, i.e. it checks the validity of a formula for all states in the model. $\mathsf{E} \vdash MC^2$ has been developed as a model checker that can easily be linked to a wide range of existing high-level modelling tools based on, for instance, stochastic process algebras, stochastic Petri nets, or queueing networks. A whole variety of such tools exists [20], most of them using dedicated formats to store the rate matrix $\mathbf{R}$ that is obtained from the high-level specification. The matrix $\mathbf{R}$, together with the proposition-labelling function L, constitutes the interface between the high-level formalism at hand and the model checker $\mathsf{E} \vdash MC^2$. Currently, $\mathsf{E} \vdash MC^2$ accepts CTMCs represented in the `tra`-format as generated by the stochastic process algebra tool TIPPTOOL [22], but the tool is designed in such a way that it enables a filter plug-in functionality to bridge to various other input formats. This is realized via JAVA's dynamic class loading capability.

3.1 The Model Checking Algorithm

Once the matrix $\mathbf{R}$ and the labelling L of a CTMC $\mathcal{M}$ have been initialised, the model checking algorithm implemented in $\mathsf{E} \vdash MC^2$ essentially proceeds in the same way as for model checking CTL [11]. For a given formula Φ it recursively computes the sets of states $Sat(.)$ satisfying the sub-formulas of Φ, and constructs the set $Sat(\Phi)$ from them. The verification of probabilistic and steady-state properties relies on the constructive characterizations for $Prob(s, \varphi)$ and $\pi_{S'}(s)$ as established in [8].

Steady-state properties. For calculating $\mathcal{S}_{\bowtie p}(\Phi)$ the tool follows a two-phase approach: first, the bottom strongly connected components (BSCC) of $\mathcal{M}$ are determined by a standard graph algorithm [28]. Then, the steady-state probability distribution is calculated for each of the BSCC. Each step requires the solution of a *linear system of equations* in the size of the BSCC. More precisely, let G be the underlying directed graph of $\mathcal{M}$ where vertices represent states and where there is an edge from s to s' iff $\mathbf{R}(s, s') > 0$. Sub-graph B is a BSCC of G if it is a strongly connected component such that for any $s \in B$, $Reach(s) \subseteq B$. We have $\pi_{s'}(s) = 0$ iff s' does not occur in any BSCC reachable from s. Let B be a BSCC of G with $Reach(s) \cap B \neq \varnothing$, or equivalently, $B \subseteq Reach(s)$, and assume that a_B is an atomic proposition such that $a_B \in L(s)$ iff $s \in B$. Then $\Diamond a_B$ is a path-formula in **CSL** and $Prob(s, \Diamond a_B)$ is the probability of reaching B from s at some time t. For $s' \in B$, $\pi_{s'}(s)$ is given by $\pi_{s'}(s) = Prob(s, \Diamond a_B) \cdot \pi_B(s')$ where $\pi_B(s')$ =1 if $B = \{s'\}$, and otherwise π_B satisfies the linear system of equations[2]

$$\sum_{\substack{s \in B \\ s \neq s'}} \pi_B(s) \cdot \mathbf{R}(s, s') = \pi_B(s') \cdot \sum_{\substack{s \in B \\ s \neq s'}} \mathbf{R}(s', s) \quad \text{such that} \quad \sum_{s \in B} \pi_B(s) = 1. \tag{1}$$

Linear systems of equations can be solved either directly (e.g. Gaussian elimination or LU-decomposition) or by iterative methods such as the power method, Jacobi iteration, Gauß-Seidel iteration and successive over-relaxation [27]. Iterative methods compute approximations to the exact result up to a prespecified precision ε. Although (except for the power method) convergence of the iterative methods is not guaranteed, this problem only appears for pathological cases in practice. The major advantage of these methods is that the involved matrices do not change during the computation (i.e. fill-in is avoided), and hence the buildup of rounding errors is nonexistent [19,27]. In addition, direct methods are known

[2] In [8] the above linear system of equations is defined in a slightly different way, by characterizing the steady-state probabilities in terms of the embedded DTMC.

to be only practical for state spaces of up to a few hundred states, while iterative methods have successfully been applied for much larger systems (up to 10^7 states) [14]. For these reasons, $\mathsf{E} \vdash MC^2$ supports all of the above mentioned iterative methods to solve (1).

Probabilistic path-formulas. Calculating the probabilities $Prob(s, \varphi)$ proceeds as in the discrete-time case [13,17,5], except for the time-bounded until that is particular to the continuous-time case. More precisely:

Next: $Prob(s, X\Phi)$ is obtained by multiplying the transition probability matrix $\mathbf{P}$ with the (boolean) vector $\mathbf{i}_\Phi = (i_\Phi(s))_{s\in S}$ characterizing $Sat(\Phi)$, i.e. $i_\Phi(s) = 1$ if $s \models \Phi$, and 0 otherwise.

Until: $Prob(s, \Phi_1 \,\mathcal{U}\, \Phi_2)$ is obtained by solving a linear system of equations of the form $\mathbf{x} = \overline{\mathbf{P}} \cdot \mathbf{x} \; + \; \mathbf{i}_{\Phi_2}$ where $\overline{\mathbf{P}}(s, s') = \mathbf{P}(s, s')$ if $s \models \Phi_1 \wedge \neg\Phi_2$ and 0 otherwise. $Prob(s, \Phi_1 \,\mathcal{U}\, \Phi_2)$ is the least solution of this set of equations. $\mathsf{E} \vdash MC^2$ computes the least solution by one of the standard methods mentioned above for the steady-state operator.

Time-bounded until: to compute the time-bounded until operator, we use the following characterization:

Theorem 1. [8] *The function $S \times I\!R_{\geqslant 0} \to [0,1]$, $(s,t) \mapsto Prob(s, \Phi_1 \,\mathcal{U}^{\leqslant t}\, \Phi_2)$ is the least fixed point of the higher-order operator $\Omega : (S \times I\!R_{\geqslant 0} \to [0,1]) \to (S \times I\!R_{\geqslant 0} \to [0,1])$ where*[3]

$$\Omega(F)(s,t) = \begin{cases} 1 & \textit{if } s \models \Phi_2 \\ \sum_{s' \in S} \mathbf{R}(s,s') \cdot \int_0^t e^{-\mathbf{E}(s)\cdot x} \cdot F(s', t-x)\; dx & \textit{if } s \models \Phi_1 \wedge \neg\Phi_2 \\ 0 & \textit{otherwise.} \end{cases}$$

This result suggests the following iterative method to approximate $Prob(s, \Phi_1 \,\mathcal{U}^{\leqslant t}\, \Phi_2)$: let $F_0(s,t) = 0$ for all s, t and $F_{k+1} = \Omega(F_k)$. Then,

$$\lim_{k\to\infty} F_k(s,t) \;=\; Prob(s, \Phi_1 \,\mathcal{U}^{\leqslant t}\, \Phi_2).$$

Each step in the iteration amounts to solve an integral of the following form:

$$F_{k+1}(s,t) = \sum_{s'\in S} \mathbf{R}(s,s') \cdot \int_0^t e^{-\mathbf{E}(s)\cdot x} \cdot F_k(s', t-x)\; dx,$$

if $s \models \Phi_1 \wedge \neg\Phi_2$. In [8], we proposed to solve these integrals numerically based on quadrature formulas with, say, $N+1$ equally spaced interpolation points $x_m = m \cdot \frac{t}{N}$ $(0 \leqslant m \leqslant N)$ such as trapezoidal, Simpson, or Romberg integration schemes. For the trapezoidal method, for instance, this amounts to approximate

$$F_{k+1}(s, x_m) \quad\approx\quad \sum_{s'\in S} \mathbf{R}(s,s') \cdot \sum_{j=0}^{m} \alpha_j \cdot e^{-\mathbf{E}(s)\cdot x_j} \cdot F_k(s', x_m - x_j)$$

where for fixed m, $\alpha_0 = \alpha_m = \frac{t}{2N}$ and $\alpha_j = \frac{t}{N}$ for $0 < j < m$. However, practical experiments with $\mathsf{E} \vdash MC^2$ revealed that these schemes may result in inaccurate results by overestimating the impact of the 'leftmost' intervals. We therefore take a different route by using piecewise integration, and approximating

[3] The underlying partial order on $S \times \mathbb{R}_{\geqslant 0} \to [0,1]$ is defined for F_1, $F_2 : S \times \mathbb{R}_{\geqslant 0} \to [0,1]$ by $F_1 \leqslant F_2$ iff $F_1(s,t) \leqslant F_2(s,t)$ for all s, t.

$$F_{k+1}(s, x_m) \approx \sum_{s' \in S} \mathbf{R}(s, s') \cdot \sum_{j=0}^{m} \int_{x_j - \beta_j}^{x_j + \beta_{j+1}} e^{-\mathbf{E}(s) \cdot x} \, dx \cdot F_k(s', x_m - x_j)$$

where $\beta_0 = \beta_{m+1} = 0$ and $\beta_j = \frac{t}{2N}$ for $0 < j \leqslant m$. Note that the resulting integrals are easily solved because they only involve exponential distributions. So, discretization is used merely to restrict the impact of possible state changes to the interpolation points $x_0, \ldots, x_N$. The influence of the number of interpolation points on the accuracy and the run-time of the algorithm is one of the interesting aspects discussed in Section 4.

As in [17] for until and time-bounded until some pre-processing is done (on the underlying graph G of CTMC $\mathcal{M}$) before the actual model checking is carried out. First we determine the set of states for which the (fair) CTL-formula $\exists(\Phi_1 \,\mathcal{U}\, \Phi_2)$ is valid, i.e. we compute $Sat(\exists(\Phi_1 \,\mathcal{U}\, \Phi_2))$. This is done in the usual iterative way [17]. For states not in this set the respective probabilistic until-formula will have probability 0. In a similar way, we compute the set of states for which the probability of these properties will be 1. This is done by computing the set of states $Sat(\forall(\Phi_1 \,\mathcal{U}\, \Phi_2))$ (up to fairness, cf. [9]) in the usual iterative way [17]. As a result, the actual computation, being it the solution of the linear system of equations in case of an unbounded until, or the solution of the system of Volterra integral equations in case of the time-bounded until, can be restricted to the remaining states. This not only reduces the number of states, but also speeds up the convergence of the iterative algorithms.

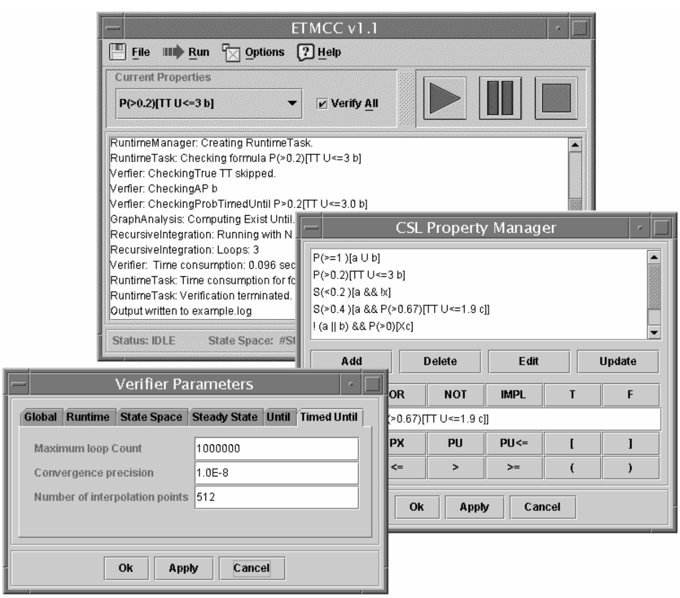


Fig. 1. User interface of $\mathsf{E} \vdash MC^2$

3.2 Tool Architecture

E ⊢ MC^2 has been written entirely in JAVA (version 1.2), an object-oriented language known to provide platform independence and to enable fast and efficient program development. Furthermore, support for the development of graphical user interfaces as well as grammar parsers are at hand. For the sake of simplicity, flexibility and extensibility we abstained from low-level optimizations, such as minimization of object invocations. The design and implementation took approximately 8 man-months, with about 8000 lines of code for the kernel and 1500 lines of code for the GUI implementation, using the SWING library. The tool architecture consists of five components, see Fig. 2.

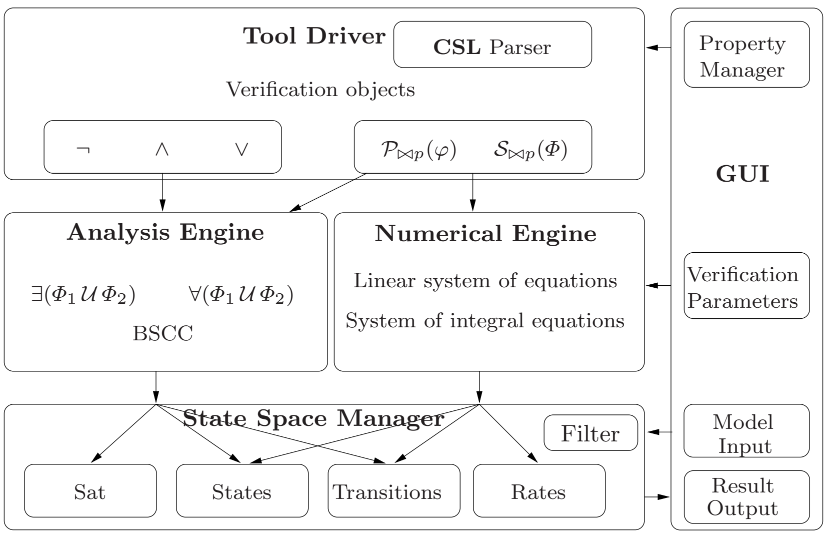


Fig. 2. The tool architecture

Graphical User Interface (cf. Fig. 1) enables the user to load a model $\mathcal{M}$, a labelling L, and the properties to be checked. It prints results on screen or writes them into file and allows the user to construct **CSL**-formulas by the '**CSL** Property Manager'. Several verification parameters for the numerical analysis, such as solution method, precision ε and number of interpolation points N, can be set by the user.

Tool Driver controls the model checking procedure. It parses a **CSL**-formula and generates the corresponding parse tree. Subsequent evaluation of the parse tree results in calls to the respective verification objects that encapsulate the verification sub-algorithms. These objects, in turn, use the analysis and/or numerical engine. For instance, checking $\mathcal{P}_{\leqslant p}(\Phi_1 \,\mathcal{U}\, \Phi_2)$ involves a pre-processing step (as mentioned above) that isolates states satisfying $\exists(\Phi_1 \,\mathcal{U}\, \Phi_2)$ and $\forall(\Phi_1 \,\mathcal{U}\, \Phi_2)$. The results of this step are passed to the numerical engine that computes the corresponding (non-zero) probabilities.

Numerical Engine is the numerical analysis engine of $\mathsf{E} \vdash MC^2$. It computes the solution of linear systems of equations and of systems of Volterra integral equations in the way explained above on the basis of the parameters provided by the user via the GUI, such as the precision ε, the 'maximum loop count' (the maximal number of iterations before termination), or the number N of interpolation points used for the piecewise integration.

Analysis Engine is the engine that supports graph algorithms, for instance, to compute the BSCC in case of steady-state properties, and standard model checking algorithms for CTL-like until-formulas. The latter algorithms are not only used as a pre-processing phase of checking until-formulas (as explained above), but they also take care of instances of the special cases $\mathcal{P}_{\geqslant 1}(\varphi)$ and $\mathcal{P}_{>0}(\varphi)$ where the numerical analysis tends to produce 'wrong' results (such as $0.99999\ldots$ rather than 1.0) due to machine imprecision.

State Space Manager represents DTMCs and CTMCs in a uniform way. In fact, it provides an interface between the various checking and analysis components of $\mathsf{E} \vdash MC^2$ and the way in which DTMCs and CTMCs are actually represented. This eases the use of different, possibly even symbolic state space representations. It is designed to support input formats of various kinds, by means of a simple plug-in-functionality (using dynamic class loading). It maintains information about the validity of atomic propositions and of sub-formulas for each state, encapsulated in a 'Sat' sub-component. After checking a sub-formula, this sub-component stores the results, which might be used later. In the current version of the tool, the state space is represented as a sparse matrix [27]. The rate matrix $\mathbf{R}$ (and its transposed $\mathbf{R}^T$) is stored, while the entries of $\mathbf{E}$ and $\mathbf{P}$ are computed on demand. All real values are stored in the IEEE 754 floating point format with double precision (64 bit).

4 Application Case Studies

In this section we report on experiences with $\mathsf{E} \vdash MC^2$ in the context of model checking two Markov chain models that have been generated from high-level formalisms, namely queueing networks and generalized stochastic Petri nets. Based on these experiments we assess the sensitivity of the model checker with respect to various parameters. We ran the experiments on a 300 MHz SUN Ultra 5/10 workstation with 256 MB memory under the Solaris 2.6 operating system. In the case studies we solve linear systems of equations by means of the Gauß-Seidel method. All recorded execution times are wall clock times.

4.1 A Simple Tandem Queue

As a first, simple example we consider a queueing network (with blocking) taken from [23]. It consists of a M/Cox$_2$/1-queue sequentially composed with a M/M/1-queue, see Fig. 3. Due to space constraints, we have to refer to [12] for a thorough introduction to networks of queues. Both queueing stations have a capacity of c

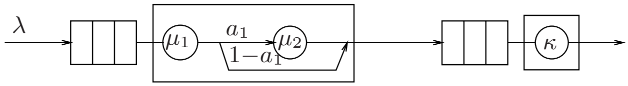


Fig. 3. A simple tandem network with blocking [23]

jobs, $c > 0$. Jobs arrive at the first queueing station with rate λ. The server of the first station executes jobs in one or two phases; that is, with probability $1 - a_1$ a job is served with rate μ_1 only, and with probability a_1, the job has to pass an additional phase with rate μ_2. Once served, jobs leave the first station, and are queued in the second station where service takes place with rate κ. In case the second queueing station is fully occupied, i.e. its server is busy and its queue is full, the first station is said to be blocked. Note that in this situation, the second phase of the first server is blocked and the first server can only pass a job that just finished the first phase to the second phase (which happens with probability a_1), but the "bypass" of the second phase is also blocked. For the experiments we take the following values for the parameters of the queue: $\lambda = 4 \cdot c$, $\mu_1 = 2$, $\mu_2 = 2$, $\kappa = 4$, and $a_1 = 0.1$. We consider $c = 2$, which amounts to 15 states and 33 transitions, $c = 5$, i.e. 66 states and 189 transitions and $c = 20$, i.e. 861 states and 2851 transitions. The following atomic propositions are considered:

– *full* is valid iff the entire tandem network is entirely populated, i.e. iff both queueing stations contain c jobs,
– *fst* is valid iff no new arriving job (with rate λ) can be accepted anymore, because the first queue is entirely populated.
– *snd* is valid iff the first queueing station is blocked, because the second queue is entirely populated.

It should be noticed that *full* characterizes a single state, and hence, for large c identifies a rare event, i.e. a situation that appears with very low probability. The following steady-state properties are checked: $\mathcal{S}_{\bowtie p}(\mathit{full})$, $\mathcal{S}_{\bowtie p}(\mathit{fst})$, and $\mathcal{S}_{\bowtie p}(\mathcal{P}_{\bowtie q}(X\, \mathit{snd}))$, for arbitrary p and q. The latter property is valid if the steady-state probability to be in a state that can reach a state in which the first queueing station is blocked in a single step with probability $\bowtie q$ satisfies $\bowtie p$. We do not instantiate p and q, as the execution times and computed probabilities will be the same for all p and q (except for the extremal cases 0 and 1); only the comparison with the bounds might lead to a different outcome. Thus, $p, q \in]0, 1[$. For the steady-state properties we vary the precision ε of the computed probability, which is a parameter to the model checker. The results are listed in Table 1.

The third column indicates the number of iterations needed to reach the result with the desired precision. Recall that the model checker checks the validity of **CSL**-formulas for all states in the CTMC.

The following probabilistic path properties are checked: $\mathcal{P}_{\bowtie p}(\diamond^{\leqslant t} \mathit{full})$, $\mathcal{P}_{\bowtie p}(\diamond^{\leqslant t} \mathit{fst})$ and $\mathcal{P}_{\bowtie p}(\mathit{snd}\,\mathcal{U}^{\leqslant t}\, \neg \mathit{snd})$. The latter property refers to the prob-

Table 1. Statistics for checking steady-state properties on the tandem queue

# states	ε	# iterations	$\mathcal{S}_{\bowtie p}(full)$ time (in *sec*)	$\mathcal{S}_{\bowtie p}(fst)$ time (in *sec*)	$\mathcal{S}_{\bowtie p}(\mathcal{P}_{\bowtie q}(X\,snd))$ time (in *sec*)
15	10^{-4}	62	0.012	0.012	0.013
	10^{-6}	107	0.016	0.017	0.016
($c = 2$)	10^{-8}	146	0.017	0.018	0.019
66	10^{-4}	77	0.028	0.028	0.065
	10^{-6}	121	0.041	0.042	0.076
($c = 5$)	10^{-8}	159	0.048	0.085	0.181
861	10^{-4}	74	0.569	0.498	1.567
	10^{-6}	118	0.644	0.643	1.935
($c = 20$)	10^{-8}	158	0.811	0.778	2.369

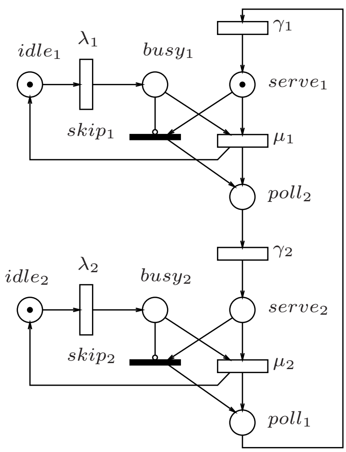
Table 2. Statistics for checking $\mathcal{P}_{\bowtie p}(\Phi_1\,\mathcal{U}^{\leqslant t}\,\Phi_2)$-formulas on the tandem queue

# states	t	# interpolation points	$\mathcal{P}_{\bowtie p}(\diamond^{\leqslant t} full)$ # iter.	time (in *sec*)	$\mathcal{P}_{\bowtie p}(\diamond^{\leqslant t} fst)$ # iter.	time (in *sec*)	$\mathcal{P}_{\bowtie p}(snd\,\mathcal{U}^{\leqslant t}\,\neg snd)$ # iter.	time (in *sec*)
15	2	64	18	2.497	11	1.045	4	0.144
		128	18	9.762	11	4.082	4	0.566
		256	18	22.19	11	16.30	4	2.248
		512	18	156.2	11	69.04	4	9.067
($c = 2$)		1000	18	602.3	11	248.6	4	34.27
15	10	64	45	6.506	12	1.140	4	0.145
		128	43	24.00	12	4.575	4	0.568
		256	43	52.85	12	17.94	4	2.309
		512	43	383.1	12	75.13	4	8.994
($c = 2$)		1000	43	1433	12	274.9	4	34.38
15	100	64	472	104.6	12	2.133	4	0.229
		128	344	284.9	12	7.682	4	0.817
		256	285	958.1	12	31.07	4	3.361
		512	260	3582	12	123.8	4	13.51
($c = 2$)		1000	252	13201	12	493.8	4	51.49
861	2	64	36	448.3	29	347.3	21	9.608
		128	36	1773	29	1336	21	38.90
		256	36	7028	29	5293	21	150.5
($c = 20$)		512	36	28189	29	21914	21	600.1

ability of leaving a situation in which the second queue is entirely populated. All path-properties are checked with precision $\varepsilon = 10^{-6}$. We vary the time-span t (over 2, 10 and 100), and the number of interpolation points for the piecewise integration from 64 up to 1000. The results for $c = 2$ are listed in the upper part of Table 2. Note the difference in computation time for the different properties. Whereas $\mathcal{P}_{\bowtie p}(snd\,\mathcal{U}^{\leqslant t}\,\neg snd)$ can be checked rather fast, calculating the probability for reaching a *fst*-state within a certain time bound, and — in particular — until reaching a *full*-state takes significantly more time. Since the CTMC is strongly connected, a *full*- or *fst*-state can (eventually) be reached from any other state, and hence for all states the probability for reaching these states within time t must be calculated. In addition, the probability of reaching the single *full*-state is low, especially for larger c, and quite a number of iterations are needed in that case to obtain results with the desired precision. Since there are several *fst*-states in the CTMC, this effect is less important for $\mathcal{P}_{\bowtie p}(\diamond^{\leqslant t} fst)$. For the last property (last two columns), probabilities need only be computed for *snd*-states rather than for all states, and precision is reached rather quickly as the real probabilities are close to 1. These effects become more apparent when increasing the state space. This is reflected by the results in the lower part of Table 2 where we considered a CTMC of almost 1000 states.

4.2 A Cyclic Server Polling System

In this section, we consider a cyclic server polling system consisting of d stations and a server, modelled as a GSPN.[4] The example is taken from [25], where a detailed explanation can be found. For $d = 2$, i.e. a two-station polling system, the GSPN model is depicted on the left. For a d-station polling system, the Petri net is extended in the obvious way. Place $idle_i$ represents the condition that station i is idle, and place $busy_i$ represents the condition that station i has generated a job. The server visits the stations in a cyclic fashion. After polling station i (place $poll_i$) the server serves station i (place $serve_i$), and then proceeds to poll the next station. The times for generating a message, for polling a station and for serving a job are all distributed exponentially with parameters λ_i, γ_i and μ_i, respectively. In case the server finds station i idle, the service time is zero which is modelled by the immediate transition $skip_i$ and the inhibitor arc from place $busy_i$ to transition $skip_i$. In this study we consider polling systems with $d = 3$, 5, 7 and 10 stations (like in [25]). The corresponding CTMCs have 36, 240, 1344 and 15360 states (84, 800, 5824 and 89600 transitions). The polling systems are assumed to be symmetric, i.e. all λ_i have the same numerical values, and the same is true for all $\gamma_i = 1$ and all $\mu_i = 200$. We set $\lambda_i = \mu_i/d$.



In the context of GSPNs, it is rather natural to identify the set of places that possess a token in a given marking — i.e. a state of our CTMC — with the set of atomic propositions valid in this state. Based on these atomic propositions, we check the following properties on the polling system: $\neg(poll_1 \wedge poll_2)$, stating that the server never polls both stations at the same time; $\mathcal{P}_{\bowtie p}(\neg serve_2 \, \mathcal{U} \, serve_1)$, i.e. with probability $\bowtie p$ station 1 will be served before station 2; $busy_1 \Rightarrow \mathcal{P}_{\geqslant 1}(\Diamond poll_1)$, so once station 1 has become busy, it will eventually be polled; $busy_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} poll_1)$, once station 1 has become busy, with probability $\bowtie p$ it will be polled within t time units. (We let $t = 1.5$.) The following steady state

[4] We refer to [1] for details on the semantics of GSPNs. In particular, the existence of immediate transitions (i.e. the black transitions) leads to so-called vanishing markings in the reachability graph which, however, can be eliminated easily. Our model checker works on the resulting tangible reachability graph which is isomorphic to a CTMC.

formulas are considered: $\mathcal{S}_{\bowtie p}(busy_1 \wedge \neg serve_1)$, which says that the probability of station 1 being waiting for the server is $\bowtie p$; and $\mathcal{S}_{\bowtie p}(idle_1)$, stating that the probability of station 1 being idle is $\bowtie p$. Like before, $p \in]0, 1[$. All path-

Table 3. Statistics for checking **CSL**-formulas on the polling system

d	# states	$\neg(poll_1 \wedge poll_2)$ time (in *sec*)	$\mathcal{P}_{\bowtie p}(\neg serve_2 \,\mathcal{U}\, serve_1)$ time (in *sec*)	$busy_1 \Rightarrow \mathcal{P}_{\geqslant 1}(\Diamond poll_1)$ time (in *sec*)
3	36	0.002	0.031	0.005
5	240	0.002	0.171	0.009
7	1344	0.005	1.220	0.011
10	15360	0.037	16.14	0.080

		$busy_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 1.5} poll_1)$		$\mathcal{S}_{\bowtie p}(busy_1 \wedge \neg serve_1)$		$\mathcal{S}_{\bowtie p}(idle_1)$	
d	# states	# iter.	time (in *sec*)	# iter.	time (in *sec*)	# iter.	time (in *sec*)
3	36	8	2.308	39	0.044	39	0.038
5	240	12	30.92	61	0.103	61	0.102
7	1344	14	308.5	80	0.677	80	0.658
10	15360	18	7090	107	11.28	107	11.29

properties were checked with precision $\varepsilon = 10^{-6}$, and the number of interpolation points for numerical integration was set to 64. The steady-state properties were checked for $\varepsilon = 10^{-8}$.

4.3 Assessment of the Tool

Verification time. From the results of our case studies we observe that checking **CSL**-formulas consisting of just atomic propositions and logical connectives is very fast. Checking steady-state properties and unbounded until-formulas is also a matter of only a few seconds, even for the 15360 state case. Measurements have shown that the performance of our tool's steady-state solution algorithm is comparable to the one of TIPPtool [22] which is based on a sophisticated sparse matrix library implemented in C. The model checking algorithm for time-bounded until $\mathcal{P}_{\bowtie p}(\Phi_1 \,\mathcal{U}^{\leqslant t}\, \Phi_2)$, which involves the approximate solution of a system of integral equations, becomes very time consuming for larger state spaces. Obviously, the execution times for checking time-bounded until strongly depend on the chosen number of interpolation points: each iteration in the piecewise integration in the worst case is of order $\mathcal{O}(N^2 \cdot K)$, where K is the number of transitions and N the number of interpolation points. In addition, the execution times depend on the arguments (i.e. Φ_1 and Φ_2) and the considered time-span (i.e. parameter t). For instance, checking $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} \Phi_2)$ involves a computation for each state (that has a non-zero and non-trivial probability of reaching a Φ_2-state), while checking $\mathcal{P}_{\bowtie p}(a \,\mathcal{U}^{\leqslant t}\, \Phi_2)$ only involves a computation for the a-labelled states (of this set). The case studies, and other experiments which we conducted showed that the main performance bottleneck of our tool is the algorithm for time-bounded until.

Accuracy of numerical results. In order to assess the numerical accuracy of the algorithm for time-bounded until, we used our tool to compute (amongst others) the cumulative distribution function of the Erlang k-distribution, that is, a convolution of k identical exponential distributions. For small k the results of the iterative algorithm are quite accurate, even for a small number of interpolation points N. The accuracy further improves as N is increased, as expected. For $k \geqslant 100$ and small N, the accuracy of the results is unacceptable, while for larger N the run-time becomes excessive.

Accuracy of verification result. Another issue — that is inherently present in all model checking approaches that rely on numerical recipes — is to avoid wrong outcomes of comparisons with a probability bound p in a sub-formula, that is then propagated upwards. Because round-off and truncation errors cannot be avoided (due to machine imprecision), this effect can happen if the computed value is very close to the bound p. For the extremal probability bounds (i.e. bounds > 0 and $\geqslant 1$), we have circumvented this problem by applying the standard model checking algorithms for $\forall$ and $\exists$ as in [17]. Furthermore we intend to use a three-valued logic such that the tool can avoid potentially wrong results, and answers 'don't know' in case some calculated (i.e. approximated) probability is within some tolerance to a probability bound p occurring in a (sub-)formula to be checked.

5 Conclusion

In this paper we have presented a model checker for (state labelled) discrete and continuous-time Markov chains. We reported on the structure of the tool, and on experiments using the model checker to verify CTMCs derived from high-level formalisms such as stochastic Petri nets and queueing networks. As far as we know, $\mathsf{E} \vdash MC^2$ is the first implementation of a bridge between such high-level specification formalisms for CTMCs and model checking.

$\mathsf{E} \vdash MC^2$ is a prototype, in particular for the moment it does not use symbolic, i.e. (MT)BDD-based, data structures. Although our own experience (and of others, cf. [16]) has shown that very compact encodings of Markov chains are possible with MTBDDs and similar data structures [23], and symbolic model checking algorithms for CTMCs do exist [8], we favor a separation of concerns: to our belief the issues of numerical stability, convergence, accuracy and efficiency are worth to be studied in isolation, without interference of the (sometimes unpredictable) effects of BDD-based computations. In addition, none of the high-level modelling tools for generating CTMCs uses BDD-based data structures, as far as we know.

Our decision to implement the model checker $\mathsf{E} \vdash MC^2$ in Java turned out to be a good choice. In particular it allowed us to develop an easy-to-use user interface along with the model checker engine. Also the numerical computations have a good performance in Java; e.g., the computation of steady-state properties is comparable to (optimised) existing C implementations. Our experiments with $\mathsf{E} \vdash MC^2$ have shown that the checking of time-bounded until-properties requires an efficiency improvement. We are currently considering alternative ways to model check this operator [7].

Acknowledgement

The authors thank Lennard Kerber (Erlangen) for his contribution to assessing the accuracy of the tool output and Christel Baier (Bonn) for her valuable contributions and discussions.

References

1. M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalised stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Tr. on Comp. Sys.*, **2**(2): 93–122, 1984. 348, 358
2. L. de Alfaro, M.Z. Kwiatkowska, G. Norman, D. Parker and R. Segala. Symbolic model checking for probabilistic processes using MTBDDs and the Kronecker representation. In *TACAS*, LNCS (this volume), 2000. 349
3. A. Aziz, V. Singhal, F. Balarin, R. Brayton and A. Sangiovanni-Vincentelli. It usually works: the temporal logic of stochastic systems. In *CAV*, LNCS 939: 155–165, 1995. 347
4. A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Verifying continuous time Markov chains. In *CAV*, LNCS 1102: 269–276, 1996. 347, 348
5. C. Baier. On algorithmic verification methods for probabilistic systems. Habilitation thesis, Univ. of Mannheim, 1999. 347, 352
6. C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *ICALP*, LNCS 1256: 430–440, 1997. 347
7. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. 2000 (submitted). 360
8. C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR*, LNCS 1664: 146–162, 1999. 347, 348, 349, 350, 351, 352, 360
9. C. Baier and M. Kwiatkowska. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Inf. Proc. Letters*, **66**(2): 71–79, 1998. 350, 353
10. I. Christoff and L. Christoff. Reasoning about safety and liveness properties for probabilistic systems. In *FSTTCS*, LNCS 652: 342-355, 1992. 347
11. E.M. Clarke, E.A. Emerson and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Tr. on Progr. Lang. and Sys.*, **8**(2): 244-263, 1986. 350, 351
12. A.E. Conway and N.D. Georganas. *Queueing Networks – Exact Computational Algorithms.* MIT Press, 1989. 348, 355
13. C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Proc. IEEE Symp. on Found. of Comp. Sci.*, pp. 338–345, 1988. 347, 352
14. D.D. Deavours and W.H. Sanders. An efficient disk-based tool for solving very large Markov models. In *Comp. Perf. Ev.*, LNCS 1245: 58–71, 1997. 352
15. L. Fredlund. The timing and probability workbench: a tool for analysing timed processes. Tech. Rep. No. 49, Uppsala Univ., 1994. 348
16. G. Hachtel, E. Macii, A. Padro and F. Somenzi. Markovian analysis of large finite-state machines. *IEEE Tr. on CAD of Integr. Circ. and Sys.*, **15**(12): 1479–1493, 1996. 360

17. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form. Asp. of Comp.*, **6**(5): 512–535, 1994. 347, 348, 352, 353, 360
18. V. Hartonas-Garmhausen, S. Campos and E.M. Clarke. ProbVerus: probabilistic symbolic model checking. In *ARTS*, LNCS 1601: 96–111, 1999. 348
19. B.R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach.* John Wiley & Sons, 1998. 351
20. B.R. Haverkort and I.G. Niemegeers. Performability modelling tools and techniques. *Perf. Ev.*, **25**: 17–40, 1996. 350
21. H. Hermanns, U. Herzog and J.-P. Katoen. Process algebra for performance evaluation. *Th. Comp. Sci.*, 2000 (to appear). 348
22. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis and M. Siegle. Compositional performance modelling with the TIPPtool. *Perf. Ev.*, **39**(1-4): 5–35, 2000. 351, 359
23. H. Hermanns, J. Meyer-Kayser and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous-time Markov chains. In *Proc. 3rd Int. Workshop on the Num. Sol. of Markov Chains*, pp. 188-207, 1999. 355, 356, 360
24. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996. 348
25. O.C. Ibe and K.S. Trivedi. Stochastic Petri net models of polling systems. *IEEE J. on Sel. Areas in Comms.*, **8**(9): 1649–1657, 1990. 358
26. B. Plateau and K. Atif, Stochastic automata networks for modeling parallel systems. *IEEE Tr. on Softw. Eng.*, **17**(10): 1093–1108, 1991. 348
27. W. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994. 348, 349, 350, 351, 355
28. R.E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. of Comp.*, **1**: 146–160, 1972. 351
29. M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. IEEE Symp. on Found. of Comp. Sci.*, pp. 327–338, 1985. 347